REMARKS

Claims 18-28, and 41-59 are in the application.

Claims 22-28 are withdrawn from consideration as being directed to non-elected inventions, and will be rejoined with the application if claim 21 is deemed allowable.

Claims 18, 20-22, 41, 43, 46-49, 53, and 59 are amended. Claims 1-17 and 29-40 are cancelled without prejudice or disclaimer as being directed to a non-elected invention.

Claims 18-21 and 41-45 are subject to examination.

AMENDMENTS

Claims 18, 20, 21, 22, 41, and 43, are amended to define the "hash" as a "self-authenticating cryptographic hash" or "cryptographic self-authentication hash".

Claims 46-49, 53 and 59 are amended to provide a "self-authenticating cryptographically processed set of identifications". Claim 47 further limits this element to comprise a cryptographic hash.

Claims 18, 21, and 22 are amended to define that the hash, or set of identifications, is formed from the random optically readable characteristics or ascertainable pattern.

ALL ISSUES PRESENTED ARE EXISTING IN THE APPLICATION

The concept of a "hash" is present in claims 18, 20, 22, 41, 43, 47 and 59 prior to the proposed amendments herein.

The concept of self-authentication is present in claims 21 and 41 prior to the proposed amendments herein.

The express use of cryptographic techniques are presented in claims 46-49, and 53.

The fact that the basis for the hash, that is, "formed" from or based on the pattern finds basis in claim 43 prior to amendment.

Claim 41 is dependent from claim 46, therefore provides antecedent basis for "cryptographic[]", "self-authenticating", and "hash", and thus clearly does not present new issues; the remaining claims are conformed to this scope.

The hash employed according to the present invention is disclosed on page 26, line 23; page 32, line 19, as well as in numerous of the references incorporated into the specification. Applicants have reviewed the specification, and note that the generation of the self-authentication code is disclosed at least on page 53, lines 13-21, page 54, lines 2-8, page 56, lines 2-8, and page 57, lines 21-26.

ADVISORY ACTION

The Examiner, in the outstanding Office Action, invited applicants to "disclose the method steps of generating a hash function in specific terms [as elements of the claims] … if it is presented in a manner that do[es] not require further search and consideration."

The advisory action dated June 22, 2006 states: "The addition of "self-authenticating cryptographic", "generated from a reading" into claim[] 18; "self-authenticating … with the storage medium" into claim 59, "cryptographic" limitation before "self-authenticating" and adding the limitation "hash" into claim 21 which would differentiate it [with the added limitation into claim 1], "ascertainment of" into claim 21 raise new issues requiring further consideration and possible search."

Applicants interpret this statement to mean that based on the art of record, the consolidation of the claim expression to present a "self-authenticating cryptographic hash" derived from the random pattern distinguishes the art. However, the insertion of the phrase "ascertainment of" in claim 21 raises new issues.

Applicant has streamlined the language which represents the source of the hash to "formed from" or "formed based on", which derives from claims 43 and 46.

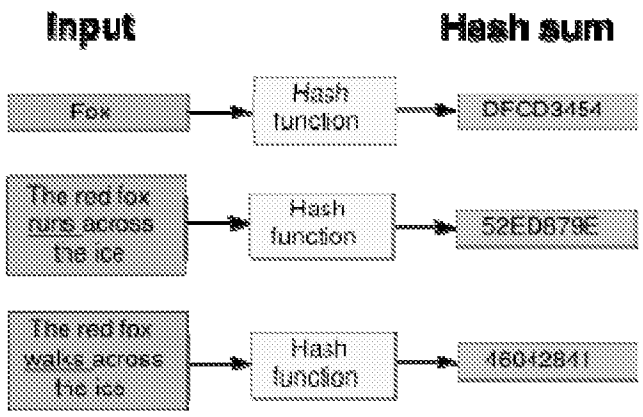
It is further respectfully submitted that, as known in the art, a hash which is self-authenticating, as provided in claim 41 prior to amendment, is inherently cryptographic, since only by hiding the key can authenticity be assured.

See Wikipedia entry for "hash function" http://en.wikipedia.org/wiki/Hash_function:

Hash function

From Wikipedia, the free encyclopedia

A **hash function** (or *hash algorithm*) is a way of creating a small digital "fingerprint" from any kind of data. The function chops and mixes the data to create the fingerprint, often called a **hash value**. The hash value is commonly represented as a short string of random-looking letters and numbers (Binary data written in hexadecimal notation). A good hash function is one that yields few hash collisions in expected input domains. In hash tables and data processing, collisions inhibit the distinguishing of data, making records more costly to find.



A typical hash function at work

Contents

[hide]

- 1 Properties of hash function
- 2 Applications of hash function
  - 2.1 Cryptography
  - 2.2 Hash tables
  - 2.3 Error correction
  - 2.4 Audio identification
  - 2.5 Rabin-Karp string search algorithm
- 3 Origins of the term
- 4 See also
- 5 References
- 6 External links

Properties of hash function

A fundamental property of all hash functions is that if two hashes (according to the same function) are different, then the two inputs are different in some way. This property is a consequence of hash functions being deterministic. On the other hand, a hash function is not injective, i.e. the equality of two hash values ideally strongly suggests, but does not guarantee the equality of the two inputs. If a hash value is calculated for a piece of data, and then one bit of that data is changed, a hash function with strong mixing property usually produces a completely different hash value.

Typical hash functions have an infinite domain, such as byte strings of arbitrary length, and a finite range, such as bit sequences of some fixed length. In certain cases, hash functions can be designed with one-to-one mapping between identically sized domain

and range. Hash functions that are one-to-one are also called permutations. Reversibility is achieved by using a series of reversible "mixing" operations on the function input.

Applications of hash function

Because *of the variety of applications for hash functions (details below), they are often tailored to the application. For example, cryptographic hash functions* assume the existence of an adversary who can deliberately try to find inputs with the same hash value. A well designed cryptographic hash function is a "one-way" operation: there is no practical way to calculate a particular data input that will result in a desired hash value, so it is also very difficult to forge. Functions intended for cryptographic hashing, such as MD5, are commonly used as stock hash functions.

Functions for error detection and correction focus on distinguishing cases in which data has been disturbed by random processes. When hash functions are used for checksums, the relatively small hash value can be used to verify that a data file of any size has not been altered.

Cryptography

Main article: cryptographic hash function

A typical cryptographic one-way function is not one-to-one and makes an effective hash function; a typical cryptographic trapdoor function *is* one-to-one and makes an effective randomization function.

Hash tables

Main article: Hash table

Hash tables, a major application for hash functions, enable fast lookup of a data record given its *key*. (Note: Keys are not usually secret as in cryptography, but both are used to "unlock" or access information.) For example, keys in an English dictionary would be English words, and their associated records would contain definitions. In this case, the hash function must map alphabetic strings to indexes for the hash table's internal array.

The generally impossible/impractical ideal for a hash table's hash function is to map each key to a unique index (see perfect hashing), because this guarantees access to each data record in the first probe into the table.

Hash functions that are truly random with uniform output (including most cryptographic hash functions) are good in that, on average, only one or two probes will be needed (depending on the load factor). Perhaps as important is that excessive collision rates with random hash functions are highly improbable—if not computationally infeasible for an adversary. However, a small, predictable number of collisions is virtually inevitable (see birthday paradox).

In many cases, a heuristic hash function can yield many fewer collisions than a random hash function. Heuristic functions take advantage of regularities in likely sets of keys. For example, one could design a heuristic hash function such that file names such as `FILE0000.CHK`, `FILE0001.CHK`, `FILE0002.CHK`, etc. map to successive indices of the table, meaning that such sequences will not collide. Beating a random hash function on "good" sets of keys usually means performing much worse on "bad" sets of keys, which can arise naturally—not just through attacks. Bad performance of a hash table's hash function means that lookup can degrade to a costly linear search.

Aside from minimizing collisions, the hash function for a hash table should also be fast relative to the cost of retrieving a record in the table, as the goal of minimizing collisions is minimizing the time needed to retrieve a desired record. Consequently, the optimal balance of performance characteristics depends on the application.

One of the most respected hash functions for use in typical hash tables is Bob Jenkins' LOOKUP2 hash function, published in an article in Dr. Dobb's Journal. The hash function performs well as long as there is no adversary, for it is trivially reversible and useless as a cryptographic hash function.

Error correction

Main article: Error correction and detection

Using a hash function to detect errors in transmission is straightforward. The hash function is computed for the data at the sender, and the value of this hash is sent with the data. The hash function is performed again at the receiving end, and if the hash values do not match, an error has occurred at some point during the transmission. This is called a redundancy check.

For error correction, a distribution of likely perturbations is assumed at least approximately. Perturbations to a string are then classified into large (improbable) and small (probable) errors. The second criterion is then restated so that if we are given H(x) and x+s, then we can compute x efficiently if s is small. Such hash functions are known as error correction codes. Important sub-class of these correction codes are cyclic redundancy checks and Reed-Solomon codes.

Audio identification

For audio identification such as finding out whether an MP3 file matches one of a list of known items, one could use a conventional hash function such as MD5, but this would be very sensitive to highly likely perturbations such as time-shifting, CD read errors, different compression algorithms or implementations or changes in volume. Using something like MD5 is useful as a first pass to find exactly identical files, but another more advanced algorithm is required to find all items that would nonetheless be interpreted as identical to a human listener. Though they are not common[*citation needed*], hashing algorithms *do* exist that are robust to these minor differences. Most of the

algorithms available are not extremely robust, but some are so robust that they can identify music played on loud-speakers in a noisy room.[citation needed] One example of this in practical use is the service Shazam. Customers call a number and place their telephone near a speaker. The service then analyses the music, and compares it to known hash values in its database. The name of the music is sent to the user (for a charge!). An open source alternative to this service is MusicBrainz which creates a fingerprint for an audio file and matches it to its online community driven database.

Rabin-Karp string search algorithm

Rabin-Karp string search algorithm is a relatively fast string searching algorithm that works in O(n) time on average. It is based on the use of hashing to compare strings.

Origins of the term

The term "hash" apparently comes by way of analogy with its standard meaning in the physical world, to "chop and mix." Knuth notes that Hans Peter Luhn of IBM appears to have been the first to use the concept, in a memo dated January 1953; the term *hash* came into use some ten years later.

In the SHA-1 algorithm, for example, the domain is "flattened" and "chopped" into "words" which are then "mixed" with one another using carefully chosen mathematical functions. The range ("hash value") is made to be a definite size, 160 bits (which may be either smaller or larger than the domain), through the use of modular division.

Since the addition of the phrase "cryptographic" to claim 41 is really only an express statement of an inherent limitation, which itself appears in other claims of record, it is respectfully submitted that no issues have been presented for examination. Further, since the search of the art must have encompassed the use of "hash" for "self-authenticating", and "cryptographic hash", it is respectfully submitted that the issues now of record are all preexisting, and the amendments should be entered.

ART REJECTIONS

Claims 18-21 and 43-59 are rejected under 35 U.S.C. § 103(a) as being obvious over Li (US 5,549,953) in view of Waters (US 5,572,589).

Applicants have previously sought to distinguish Li based on the fact that a hash or cipher is not taught thereby. As previously argued, Li does not disclose any cryptographic technique, and thus not a self-authenticating cryptographic technique. "Cryptography" means secret writing. A cryptographic technique is one which exploits a "secret" to communicate information whose comprehensibility is masked. See, http://en.wikipedia.org/wiki/Cryptography.

Waters provides authentication based on a deterministic pattern (i.e., one precisely defined by the manufacturing method to the relevant extent, and the security is dependent on conformance with this predetermined structure), which is distinct from a non-deterministic pattern (e.g., resulting from uncontrolled manufacturing variations and thus random and not predetermined).

It is therefore respectfully submitted that the present claims are patentable, and a Notice of Allowance is respectfully solicited.

Respectfully submitted,
MILDE & HOFFBERG, LLP

By
Steven M. Hoffberg
Reg. No. 33,511

MILDE & HOFFBERG, LLP
10 Bank Street - Suite 460
White Plains, NY 10606

914-949-3100